2015 IEEE/RSJ International Conference on Robot Systems (IROS)

# Probabilistic Progress Prediction and Sequencing of Concurrent Movement Primitives

Simon Manschitz[1,2], Jens Kober[3], Michael Gienger[2], and Jan Peters[1,4]

***Abstract*— Classical approaches towards learning coordinated movement tasks often represent a movement in a sequential and exclusive fashion. Introducing concurrency allows to decompose such tasks into a number of separate sequences, for instance for two different end-effectors. While this results in a compact and generic representation of the individual movement primitives (MPs), it is a hard problem to learn their temporal and causal organization. This paper presents a concept for learning movement tasks that require the coordination of several controlled effectors of a robot. We firstly introduce a concept to learn and estimate the progress of individual MPs from a low number of demonstrations. Secondly, we propose a representation of the task that incorporates several concurrent sequences of MPs. Combining these two elements allows to learn and reproduce coordinated bi-manual movement tasks robustly. The synchronization of the concurrent MPs is achieved implicitly using the progress prediction. The approach is evaluated in two simulation studies with a 25 degrees of freedom two-arm robot performing a pick-and-place task.**

## I. Introduction

Imagine the problem of teaching an anthropomorphic robot the task of bi-manually slicing a block of cheese. The task can be decomposed into three sub-tasks: Grasping a knife with one hand, fixing the cheese on the plate with the second hand and finally slicing it. If the sub-tasks are performed in a purely sequential way, we have to make several decisions. Firstly, we need to decide for a sequential order of the preparatory steps "holding" and "grasping", even though the order doesn't really matter for the task. Secondly, we need to represent the coordinated parts of the task ("slicing" while "fixing") into one complex movement primitive (MP), which is rather specific and therefore does not generalize well to different situations. If the sub-tasks are instead performed concurrently, "fixing" and "grasping" are independent MPs, and can be carried out with each end effector independently and at the same time. Thus, introducing concurrency leads to a) a shorter execution time, and b) a more flexible overall behavior for this task. However, slicing the cheese requires some kind of synchronization. If the fixing hand doesn't push on the cheese, it will slip away when slicing it. The task therefore requires a synchronization of "fixing" and "slicing" in order to successfully continue the task.

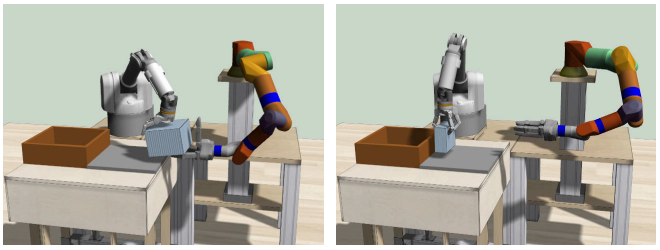


Fig. 1: The system is supposed to learn how to sequence concurrent movement primitives. We evaluate our approach on a bi-manual task where a robot has to pick up a box. Depending on the size of the box, the system has to use different strategies for picking it up.

In this paper, we focus on the class of problems with exactly these properties. Having several independent flows of MPs, a synchronization is needed at distinct points in the overall sequence. The novelty of our contribution is to learn from demonstrations when to activate which MP in the concurrent sequences in order to reproduce the task. Concurrency in this context means that the MPs controlling the robot's effectors can be activated independently and concurrently at each point in time. In contrast to other approaches in the field of bi-manual manipulation, concurrency is not only limited to the robot's end effectors. Instead, for example, the position and orientation of one end-effector could also be formulated as concurrent sequences. Also a coordination of eyes and hands could be learned. Concurrency therefore allows to decompose the task into smaller and more compact MPs, which are more intuitive and easy to reuse.

The key concepts of our approach are two-fold. Firstly, we propose a probabilistic model for learning and estimating the progress of each MP. The parameters of this progress estimation are learned from a low number of demonstrations of the task. It is an important property of our approach that the progress estimator is learned from a feature set influenced by the overall state of the system. These features are composed of pre-processed sensor information and are influenced by all MPs. During reproduction, the learned estimator maps this feature vector to a scalar progress coordinate for each MP. In our framework, the estimated progress is not necessarily a representation of the internal phase of a MP, which determines how close a MP is to it's goal position or how long it has been activated. Instead, the conditioning on a feature set allows the progress to be decoupled from

[1]S. Manschitz and J. Peters are with the Institute for Intelligent Autonomous Systems, Technische Universität Darmstadt, 64289 Darmstadt, Germany, manschitz@ias.tu-darmstadt.de, mail@jan-peters.net

[2]S. Manschitz and M. Gienger are with the Honda Research Institute Europe, 63073 Offenbach, Germany, michael.gienger@honda-ri.de

[3]J. Kober is with the Delft Center for Systems and Control, Delft University of Technology, 2628 CD, Delft, The Netherlands, j.kober@tudelft.nl

[4]J. Peters is with the Max Planck Institute for Intelligent Systems, 72076 Tübingen, Germany

the actual motion. For example, the progress might increase only if a certain event happens, even though the MP's goal position has already been reached. The second key concept is that we represent the demonstrated task as a set of concurrent sequences of MPs. The transition between consecutive MPs of each sequence is determined by interpreting the progress of an individual MP as entry and exit probability. We propose a transition indicator that is composed of the current MPs exit probability and the next MPs entry probabilities. Combining both progress prediction and concurrent sequence representation allows to reproduce a demonstrated task. The concurrent sequences are synchronized with each other only implicitly, as the individual MP entry and exit probabilities are conditioned on the global feature state influenced by all MPs. Our approach is also able to learn structural variations in the task, for instance performing a task with one or two hands depending on the feature values. The presented approaches are evaluated in two simulation studies in a pick and place task. To evaluate the experiments as realistic as possible, we utilize a physics simulation engine.

### A. Related Work

The learning from demonstration (LFD, [1]) paradigm has received a lot of attention in robotics research over the past years. Demonstration data can be acquired via kinesthetic teaching or by tracking human movements, e.g., via motion capturing or computer vision systems. The recorded data can be used to learn movements directly or for bootstrapping the learning process, e.g., by using it as prior knowledge for a reinforcement learning method. A movement can be interpreted either as a single complex movement or as a sequence of simpler movements. In literature, those simple movements are often referred to as movement primitives (MPs) [2], [3]. There are various MP representations available, including dynamic movement primitives (DMPs, [3]), probabilistic movement primitives (PROMPs, [4]), or based on Gaussian mixture models [5]. Often, MPs encode a desired trajectory in joint or task space and are activated exclusively. This concept can also be extended to bi-manual movements. Here, a single MP has to encode the desired trajectory of both end-effectors. One advantage of such an encoding is the straightforward use of existing methods. The main disadvantage is that the resulting MPs can be rather complex and are hardly reusable in different situations. Therefore, an alternative approach is to activate MPs concurrently. We argue, that if movements of two end-effectors (or of two independent control variables in general) are represented independently, the representations become simpler and are more intuitive. The drawback is, that the coordination is more difficult, as the system has more possibilities for activating the MPs. Luksch et al. [6] introduced a framework where MPs can be activated concurrently without any restrictions. The MPs are coordinated using recurrent neural networks. Due to the various possible combinations of MPs, learning the sequences is a hard problem. Therefore, the parameters of the network had to be defined by hand. A stronger coupling of the concurrent movements can be achieved, for example, by using DMPs and adding a coupling term to the original formulation [7], [8], [9]. In general, such a coupling term allows for conditioning a DMP on external signals. For coupling concurrent movements, one DMP can be conditioned for example on the state of another DMP.

The representation of a sequence is often independent of the underlying MP representation. Most concepts are based on a hierarchical architecture. Usually, the hierarchy consists of two layers, where the upper-layer sequencing layer modulates the lower-layer MPs. Kulić et al. [10] represent MPs with hidden Markov models and sequences in a motion primitive graph. The transition probabilities between consecutive MPs depend on how often the transition has been observed in the demonstrations. A movement is then generated by sampling MPs from the graph. Pastor et al. [11] encode not only the desired trajectory with DMPs, but also the expected sensor traces. As soon as a MP approaches its goal position, the most likely successor can be found by nearest neighbor classification using the current sensor signal and the expected sensor traces. The same method for finding the most likely successor is also used by Niekum et al. [12]. Here, the sequential layer is represented using a finite state machine (FSM) and learned with a beta process auto-regressive hidden Markov model, which also segments the demonstrations automatically. The approach is similar to that of Butterfield et al. [13], which use a hierarchical Dirichlet process hidden Markov model. Another possibility for representing a sequence are Petri nets [14].

Instead of having either pure sequential or concurrent MPs, it is also possible to have master and slave systems. For bi-manual tasks, for example, the movements of the second end-effector can be conditioned on the movements of the first end-effector. In that case, the movements of the first end-effector can be learned independently of the second end-effector. For example, Maeda et al. [15] condition the movements of an end-effector on the movements of a human co-worker for learning collaborative tasks.

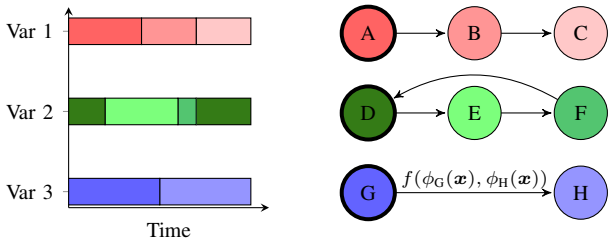
### B. Proposed Approach

We adopt the idea of concurrent MPs. In contrast to other approaches on bi-manual manipulation, the concurrency is not limited to the movements of the two end-effectors. Instead, we allow an arbitrary number of control variables. In the later examples, these are the position and orientation of each end-effector, as well the finger angles of each hand. For each control variable, one MP is active at the same time. The goal is to learn from demonstrations when to activate which MP. The novelty of our approach is that we have concurrent MP sequences that are implicitly synchronized. The concurrency keeps the MP and task descriptions simple, while the implicit synchronization ensures that the MP activations of the concurrent sequences are still conditioned on each other.

The sequence of MP activations for each control variable is represented independently with a graph. For each node in a graph, the entry and exit probabilities are learned from a global feature set, which represents the state of the entire

**Algorithm 1** Graph generation

**Require:** Features $\boldsymbol{X}^{(M)}$, Labels $\boldsymbol{y}^{(M)}$
g = graph(); // Empty Graph
**for** $m = 1 : M$ **do**
  **if** !g.hasNode($y_1^{(m)}$) **then**
    $node$ = g.addNode($y_1^{(m)}$); // Label of first point
  g.setInitialNode($y_1^{(m)}$, $true$); // Mark as initial node
  $node$.startNewTrajectory(); // Indicate new trajectory starts
  $node$.addData($\boldsymbol{X}_1^{(m)}$); // Add single data point
  **for** $i = 2 : n_m$ **do**
    **if** $y_i^{(m)} \neq y_{i-1}^{(m)}$ **then**
      **if** !g.hasNode($y_i^{(m)}$) **then**
        $node$ = g.addNode($y_i^{(m)}$);
        **if** !g.hasTransition($y_{i-1}^{(m)}, y_i^{(m)}$) **then**
          g.addTransition($y_{i-1}^{(m)}, y_i^{(m)}$);
      $node$.startNewTrajectory();
    $node$.addData($\boldsymbol{X}_i^{(m)}$);
**return** g; // Return graph




(a) Active MPs (b) Sequence Graphs

Fig. 2: Labels (active MPs) and resulting sequence graphs for three different control variables (Var). Each MP is represented by a different background color. Initial nodes are indicated by thick borders. A graph represents the sequence of MP activations of one control variable and is independent of all other graphs on this layer. The transition behavior is determined by the predicted progress $\phi$ of the current node and its successors. As the progress of each node depends on the same global feature state $\boldsymbol{x}$, the MPs are still implicitly coupled.

system. Therefore, even though the MPs of each control variable can be activated concurrently, the transition behavior is implicitly conditioned on the state of the entire system.

In this paper, a MP is a first-order dynamical system (DS) with a linear attractor behavior. Each DS has a goal in task space coordinates that is reached if the MP is activated. Depending on the control variable, a goal can be a desired position or orientation of a robot body, joint angle, force or a combination thereof and can be defined relative between bodies using reference frames. In previous work, we showed how the attractor goals and reference frames can be extracted from kinesthetic demonstrations [16], [17]. For the remainder of the paper, we therefore impose the following assumptions: We assume to know the number of concurrent sequences. We also assume that the concurrent sequences are represented in independent task spaces, so that they are not competing against each other. Even though non-independent task spaces of MPs are an interesting research problem, they are out of the scope of this paper. Finally, we assume that a previous stage the utilized MPs have been learned at and that the training data has been labeled with the active MPs over time.

The remainder of this paper is organized as follows. In Section II, the graph framework is introduced. In Section III, we show how the transition behavior between MPs can be learned by predicting their progress. The approach will be evaluated with a simulation study where different boxes have to be picked up bi-manually in Section IV. Finally, a conclusion and outlook on future work are given in Section V.

## II. Concurrent Sequence Graphs

In this section, we show how a sequence of MPs is represented using a graph, which we call sequence graph. In a sequence graph, every node is linked to a MP. During reproduction, the graph determines which MP may be activated next. Within our concurrent MP framework, one graph is used for representing the sequence of activated MPs for each control variable. We assume that a task has been demonstrated $M$ times, resulting in a data set $\boldsymbol{X}^{(m)} \in \mathbb{R}^{d \times n_m}$ and a set of labels $\boldsymbol{Y}^{(m)} \in \mathbb{R}^{c \times n_m}$. Here, $m$, $d$, $c$, and $n_m$ indicate the demonstration, dimension of the feature space, number of control variables and the number of data points for demonstration $m$, respectively. The labels indicate the active MPs for each control variable and time step. The recorded data represents the state $\boldsymbol{x} \in \mathbb{R}^{d \times 1}$ of a hand-crafted feature set for each point in time. The feature state can be comprised of any measurable magnitude, for example, external sensor signals or the joint angles of the robot. As the feature state is used for synchronizing the concurrent MPs, it is important that it represents the state of the entire system. Each graph is build separately. Therefore, the vector $\boldsymbol{y}^{(m)}$ will be used as notation in the following for the labels of a single graph.

The pseudo code for the generation of a single graph is shown in Algorithm 1. A graph is initialized with a single node and without any transitions. The node is marked as initial node and linked to the first activated MP of a sequence. For each observed transition between MPs in $\boldsymbol{y}^{(m)}$, one transition is added to the graph if it does not exist yet. If an observed MP is not yet linked to a node in the graph, the corresponding node is created first. During construction, the data is split according to the labels and assigned to the nodes. Thus, each node is linked to the data that was recorded when the corresponding MP was active during the demonstration. The data linked to one specific node will be referred to as $\tilde{\boldsymbol{X}}^{(j)}$. Here, $j \in \{1, \ldots, k\}$, where $k$ is the number of times the node was activated during the demonstrations. The data will be used for the learning the progress prediction, as shown in the following section. Fig. 2 depicts the labels and resulting sequence graphs for a demonstration of a task with three different control variables. If multiple demonstrations have been performed, there may be more than one initial node within a single graph. In this paper, each graph contains one node for each activated MP and one transition for each observed pair of succeeding MPs. For more sophisticated sequential graph structures, the interested reader may have a look at our previous work [17].

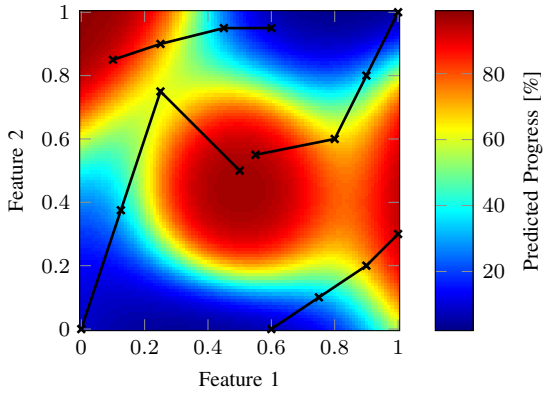



Fig. 3: Example for the progress prediction with synthetic data in a two-dimensional feature space. Four support points are chosen per trajectory, indicated by the black crosses. The trajectories start in the blue and end in the red area. The background color depicts the predicted progress for the entire feature space.

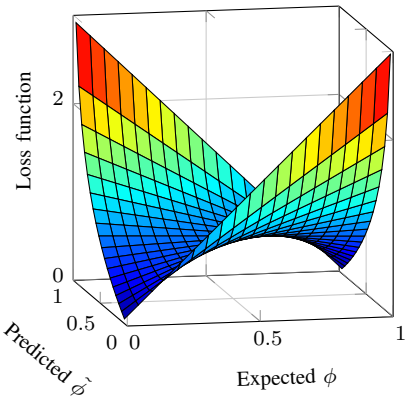



Fig. 4: Loss function for the entire range of expected and predicted progress values for a single data point. The plot indicates that the minimum is reached if prediction and expectation are equal and also, that data points close to the interval borders have the biggest impact on loss function.

## III. Progress Prediction

In this section, we show how the demonstration data can be used for learning to predict the progress of a MP. The progress $\phi$ is a real number between zero and one and is used to transition between MPs during reproduction of a task. Ideally, the predicted progress of a MP is zero when it gets activated and a transition to a successor is triggered if the progress approaches one. Basis for the learning is the data which was assigned to the MPs node during the construction of the graph. For each activation $j$ and time point $i$, an expected progress of a MP can be computed according to

$$\phi_i^{(j)} = \phi_{i-1}^{(j)} + \left\| \tilde{\boldsymbol{X}}_i^{(j)} - \tilde{\boldsymbol{X}}_{i-1}^{(j)} \right\|_1, \quad \phi_0^{(j)} = 0, \tag{1}$$

which is basically a line integral over the feature space trajectory. Subsequently, $\phi$ is normalized, so that the last $\phi$ of one activation equals one. The line integral ensures that the expected progress increases proportionally to the feature difference between two successive data points. Thus, the expected progress is independent of time (unless the time is included in the feature set), as it does not increase if the features are not changing. Based on the data and the expected progress, the goal is to learn a function, which maps the current feature state $\boldsymbol{x}$ onto the desired interval $[0, 1]$. Mapping an input to a continuous valued output is usually treated as regression problem in machine learning. Typical regression methods are not directly applicable here, as their output can be any real number. Therefore, they require an additional stage, which maps a real number onto the desired interval. Such a mapping could be achieved with a Sigmoid function or by clipping. Still, it is not obvious which method to choose, as it is for example not clear whether an output value larger than one should trigger a MP transition or not.

Our method does not require an additional stage, as it inherently maps the feature state onto the interval $[0, 1]$. The key idea is, that the progress can be also interpreted as an exit and entry probability of a node as follows

$$p_{\text{exit}} = \phi, \tag{2}$$

$$p_{\text{entry}} = 1 - \phi. \tag{3}$$

Learning both probabilities can be achieved by formally treating the progress prediction as binary classification problem. The difference to usual classification problems is, that we do not use the usual 1-of-$K$ coding scheme where class labels are either one or zero. Instead, the expected progress is used directly as label. Thus, the binary state of the labels is extended to a continuous space where a label can be anything between zero and one. As we are not interested in the actual class labels but rather in the class probabilities, the classifier can also be seen as a regression method in our case.

We use kernel logistic regression (KLR) for learning, but in general any probabilistic binary classifier can be used [18]. The progress predicted by the KLR model is

$$\tilde{\phi}_i = \frac{1}{1 + \exp\left(\sum_{s=1}^{S} w_s k(\hat{\boldsymbol{X}}_s, \tilde{\boldsymbol{X}}_i)\right)}, \tag{4}$$

where $w_s$ is an element of the parameter vector $\boldsymbol{w} \in \mathbb{R}^{S \times 1}$, $\hat{\boldsymbol{X}}_s$ is a support point and $k$ is a kernel function applied to a support point and the current data point $\tilde{\boldsymbol{X}}_i$. We use radial basis functions as kernels, so that $k(\hat{\boldsymbol{X}}_s, \tilde{\boldsymbol{X}}_i) = \exp(-\gamma \|\hat{\boldsymbol{X}}_s - \boldsymbol{X}_i\|^2)$. In the standard formulation of the KLR model, all data points are also chosen as support points. The resulting large number of free parameters often leads to overfitting. We therefore suggest to distribute the support points equidistant along the trajectories and to scale their number proportionally to the final value of $\phi$ before normalizing it to one. The value is a good indicator for the non-linearity of a trajectory (if the features have similar magnitudes or are scaled to a fixed range) and comes at no further cost, as it has to be computed before. Additionally, scaling the number proportionally to the non-linearity of a trajectory avoids having too few support points, which may decrease the prediction performance. In general, any sparse kernel method can be used to adjust the number of support points (e.g., [19]).

The model parameters are learned by minimizing the loss function $l$, which is the negative log-likelihood

$$l_{\text{NLL}} = -\sum_{i=1}^{N} \phi_i \log(\tilde{\phi}_i) + (1-\phi_i)\log(1-\tilde{\phi}_i). \quad (5)$$

Here, $N$ is the total number of data points assigned to the node, $\phi$ is the expected normalized progress computed with (1) and $\tilde{\phi}$ is the predicted progress by the KLR model. Note that even if a data point is not used as support point, it still has an impact on the loss function (5). Fig. 3 shows four example trajectories and the predicted progress in a two-dimensional space.

Two important properties of using continuous class labels are worth emphasizing here. Firstly, minimizing (5) is still a convex problem. Thus, the parameters can be found straightforwardly, for example, with iterative reweighted least squares [20]. Secondly, the loss function becomes minimal only if prediction and expectation are both close to either zero or one, as depicted in Fig. 4. Still, for a given expectation the minimum is achieved if prediction and expectation are equal. The slope also indicates, that data points where either expectation or prediction are close to the interval border have the biggest impact on the cost function. This insight is important, as it is a desired property. As the predicted progress is used for transitioning between MPs, the prediction has to be more accurate at the start and end of a MP activation, for avoiding premature, too late or incorrect transitions.

When reproducing a task, for each graph the node with the highest entry probability among all initial nodes is activated first. During execution, a transition probability is computed for each possible successor of the current node by multiplying the exit probability of the current node with the entry probability of the successor. If a transition probability is above a certain threshold ($0.65$ for our experiments) and has not improved over the last $50ms$, the system starts to activate the most likely successor. The values have been found via experimental validation.

## IV. Evaluations and Experiments

For evaluating our approach, we perform a simulation study where a two-armed robot has to pick up boxes with varying sizes. After picking it up, it has to be lifted up and put into a container. The simulation setup is shown in Fig. 5. The robot has two arms and in total 25 degrees of freedom (DOF). In the following, we will refer to the both robot arms as the left and the right arm. Each arm has seven DOF, while the hand of the left and right arm have four and seven DOF, respectively. For performing the task, the system has to activate the MPs for the six independent control variables correctly. As already mentioned, the control variables are the position and orientation of each end-effector and the fingers angles of each hand. The width and depth of the box are varied randomly. For performing the task, the system has to use three different strategies, depending on the size of the box (e.g., grasp it bi-manually if the box is big). The strategies are also depicted and explained in Fig. 5. The initial position of the robot is always the same.

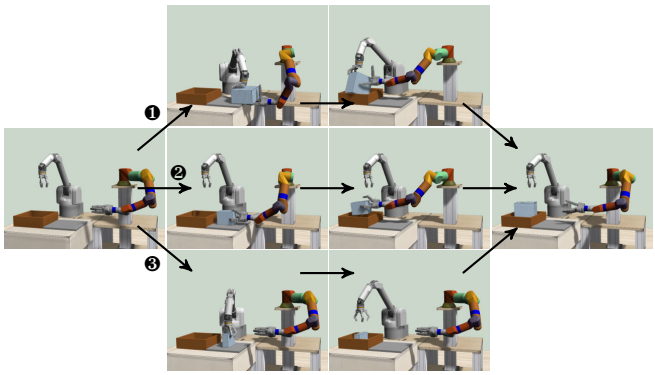



Fig. 5: Task flow for the experiment (from left to right). Different strategies are needed for achieving the task. The robot starts in its initial position. Subsequently, a box is placed on the gray plate. Width and depth of the box are set randomly. Depending on the size of the box, either both end-effectors ❶ or only the right ❷ or left end-effector have to be used ❸ for grasping the box. Subsequently, the box has to be lifted and put into a container. Finally, the robot returns to its initial position.

We manually define a total of 20 MPs for the task. Seven MPs control the position of an end-effector in world coordinates. One MP controls the left end-effector relative to the right end-effector. This MP controls the left arm when lifting the box bi-manually. Four MPs control the finger angles of a hand, respectively and two MPs control the orientation of the corresponding end-effector, respectively. For the simulation, the Vortex physics engine is utilized. No additional sensor noise or execution inaccuracies have been taken into account. The feature set is comprised of the size and position of the box and the positions of the end-effectors in world coordinates. Additionally, for each MP controlling either the orientation (relative to the world) of an end-effector or the finger angles of a hand, one feature $f$ is computed according to the equation

$$f = 1 - \exp(-0.5\boldsymbol{\Delta}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\Delta}). \quad (6)$$

Here, $\boldsymbol{\Delta}$ is a difference measure between the current state of the robot in task space coordinates and the MPs attractor goal. For the finger angles, the difference between state and goal is used, while for the orientations, the angle error is computed. The diagonal matrix $\boldsymbol{\Sigma}$ can be used to shape the feature around the attractor goal [6]. In general, our approach works with arbitrary features. However, the advantage of (6) is that the feature is intrinsically in the range $[0, 1]$, which makes further data scaling superfluous. Additionally, there is no danger of flipping signs for the orientations (e.g., jumps from $-\pi$ to $\pi$). In total, the dimension of the feature space is 24. The positions and box sizes have been scaled, so that they are also roughly in the range of $[0, 1]$.

### A. Evaluation of Progress Prediction

In a first experiment, the progress prediction is evaluated. First, 15 demonstrations are performed, five with each different strategy. For the demonstrations, we predefine the

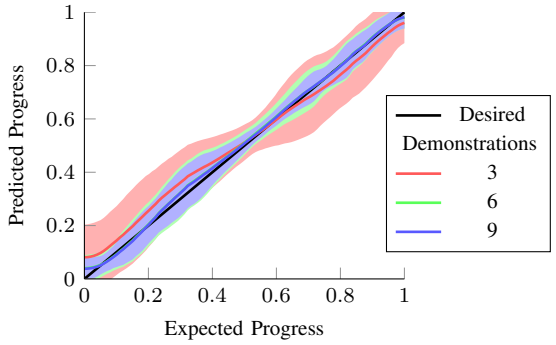


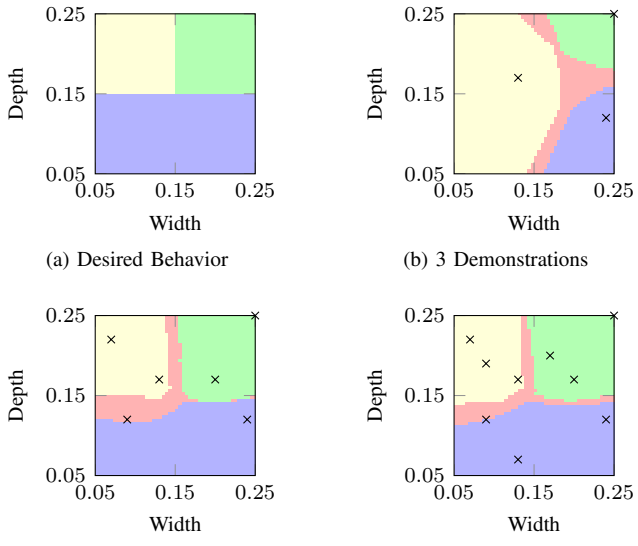

Fig. 6: Predicted and expected progress (ground truth), averaged over all MP activations and five demonstrations. Ideally, the prediction would be a straight line with slope one (black line). The background colors show the hull of two standard deviations. The box sizes used in the demonstrations are chosen randomly and therefore different than during the training phase of the progress predictors.

sequences using finite state machines and the transition behavior between MPs using thresholds. The demonstrations are split into a training set (nine demonstrations) and a test set (six demonstrations). Subsequently, the parameters of the progress predictors are learned using the training set. For the experiments, the kernel width of the RBF kernels is set to 4.0. After the training, the test set is used to evaluate the progress prediction, as depicted Fig. 6. The results show, that the improvement in the prediction accuracy is small for more than six demonstrations. Furthermore, the minimum of the standard deviation is reached at the start, center and end of the trajectory. The reason is that for the experiments the support points where chosen equidistant in feature space, and their minimum number was set to three. Therefore, the accuracy increases in the vicinity of the start, center and end of a trajectory. Even though the progress prediction works well for this task, we recommend to use a feature reduction method before training if the feature set is high-dimensional or contains many irrelevant features.

### B. Evaluation of Concurrent Framework

In a second experiment, the overall performance of the system is evaluated by using the progress prediction for transitioning between the MPs, according to Sec. III. A successful reproduction requires a transition to the correct MPs at the correct states. The MPs that are activated first during a demonstration vary depending on the size of the box (e.g., if the box is small, the left arm is not moving at all, while it has to approach the box for the other two cases). Therefore, each sequence graph has multiple initial nodes and at the start of a reproduction, the system has to choose the correct ones. The chosen initial nodes is what we call a strategy. As the initial position of the robot is always the same, the chosen strategy depends only on the size of the box. Fig. 7 depicts the different strategies chosen by the system.

If the system chooses the correct initial nodes, the task is reproduced until it is completed or a transition to an incorrect MP is triggered. Table Ia summarizes the results




(a) Desired Behavior (b) 3 Demonstrations

(c) 6 Demonstrations (d) 9 Demonstrations

Fig. 7: Actual and desired initial strategy depending on the size of the box. In the green area, the box has to be grasped bi-manually. In the yellow area, the right arm has to grasp it and in the blue area the left arm. In the red area, the system chooses a movement which was not demonstrated at all (e.g., it tried to grasp a box with a wrong orientation). Crosses mark the box sizes seen in the demonstrations.

| D | W | I | C |
|---|---|---|---|
| 3 | 3/20 | 4/20 | 13/20 |
| 6 | 3/20 | 2/20 | 15/20 |
| 9 | 2/20 | 2/20 | 16/20 |

(a) Reproduction Summary

| D | $\mu$ | $\sigma$ |
|---|---|---|
| 3 | 0.13 | 0.08 |
| 6 | 0.07 | 0.04 |
| 9 | 0.06 | 0.03 |

(b) Mean Distance Error

TABLE I: Table (a) summarizes the results of the experiments. For three, six and nine demonstrations, the percentages of wrong initial movements (W), incomplete (I) and complete (C) reproductions are given. A reproduction is incomplete if the system chooses a correct initial movement, but triggers a transition to a wrong MP during execution. Table (b) shows the mean and standard deviation of the Euclidean distances between the desired and triggered states in feature space for all complete reproductions.

of the experiments for the reproduction with 20 random box sizes. For successful reproductions, additionally the Euclidean distances between the states where transitions are triggered and the desired states are measured in feature space. Table Ib summarizes the measured distances. When training the system with the entire training set of nine demonstrations, the system is able to reproduce the task in 18 of the 20 cases. In two cases, the task could not be reproduced successfully, even though the initial nodes were chosen correctly.

### C. Discussion

The results of Table Ia show that the system sometimes performs a movement which was not demonstrated at all. The reason is the loose coupling between the graphs. There may be cases where the entry probabilities of two nodes in more than one graph are similar. The system then picks the highest entry probability for both graphs, which may lead to an incorrect behavior. As an example, consider a task where a robot has to grasp two different cups, both requiring a different grasp and a different orientation of the

end-effector. If the task is demonstrated with these two cups and then reproduced with a cup which is similar to both cups, all entry probabilities will be quite similar. The system then may for instance pick the grasp of the first cup and the orientation of the second cup. Such mixtures of demonstrated movements are undesired in our case, but may also be useful for other tasks. Therefore, we plan to couple the graphs stronger and dynamically in the future. A stronger coupling can be for example achieved by learning the MP activations globally from the predicted progress of all MPs. The goal is to combine the strength of concurrent and purely sequential MP frameworks. While concurrent MPs are simpler and easier to reuse, sequential frameworks are more predictable due to their deterministic behavior. Instead of coupling the graphs stronger in general, another possibility is to learn a dynamic coupling factor. Such a factor reflects the fact that a tasks consists of phases where concurrent movements have to be coordinated more accurately and phases where they are more independent.

For two trials, the reproduction failed even though the chosen strategy was correct (for nine demonstrations). Most often, the reason for failing was a decrease of the exit probability of a current movement, caused by a MP transition in a different graph. The decrease of the exit probability sometimes led to a deadlock, as the threshold for transitioning to a new movement was not reached. Solving such cases requires a better synchronization between the concurrent movements, which is another reason for the stronger coupling between the graphs. Additionally, the progress prediction is conditioned on the entire feature set at the moment. The prediction might benefit from a feature selection (or extraction) method, as it could reduce the dependency on irrelevant features that are influenced by transitioning to a new movement in a different graph.

## V. Conclusions and Future Work

In this paper, we proposed a task representation which incorporates several concurrent MP sequences into one consistent framework. For the individual MPs, we introduced a concept for learning and estimating their progress. Combining both concepts allows to reproduce coordinated bi-manual movement tasks robustly. The synchronization of the concurrent MPs is achieved implicitly using the progress prediction. We evaluated the approach in two simulation studies with a 25 degrees of freedom two-arm robot performing a pick-and-place task.

The results showed that the proposed approach can effectively learn a complex bi-manual task from very few demonstrations. In future work, we will investigate how to couple the graphs dynamically without loosing the benefits of a concurrent system in more detail. Additionally, we will perform experiments with the real robot and use the progress prediction for generating co-articulated movements.

## References


[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robot. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, 2009.

[2] T. Flash and B. Hochner, "Motor primitives in vertebrates and invertebrates," *Current Opinion in Neurobiology*, vol. 15, no. 6, pp. 660 – 666, 2005.

[3] S. Schaal, S. Kotosaka, and D. Sternad, "Nonlinear dynamical systems as movement primitives," in *IEEE-RAS Int. Conf. Humanoid Robots*, 2000.

[4] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems*, 2013.

[5] S. Calinon, Z. Li, T. Alizadeh, N. G. Tsagarakis, and D. G. Caldwell, "Statistical dynamical systems for skills acquisition in humanoids," in *IEEE Int. Conf. Humanoid Robots*, 2012.

[6] T. Luksch, M. Gienger, M. Muehlig, and T. Yoshiike, "Adaptive movement sequences and predictive decisions based on hierarchical dynamical systems," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2012.

[7] T. Kulvicius, M. Biehl, M. J. Aein, M. Tamosiunaite, and F. Wörgötter, "Interaction learning for dynamic movement primitives used in cooperative robotic tasks," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1450 – 1459, 2013.

[8] A. Gams, B. Nemec, A. Ijspeert, and A. Ude, "Coupling movement primitives: Interaction with the environment and bimanual tasks," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 816–830, 2014.

[9] J. Umlauft, D. Sieber, and S. Hirche, "Dynamic movement primitives for cooperative manipulation and synchronized motions," in *IEEE Int. Conf. Robotics and Automation*, 2014.

[10] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *Int. J. Rob. Res.*, vol. 31, no. 3, pp. 330–345, 2012.

[11] P. Pastor, M. Kalakrishnan, L. Righetti, L. Righetti, and S. Schaal, "Towards associative skill memories," in *IEEE-RAS Int. Conf. Humanoid Robots*, 2012.

[12] S. Niekum, S. Osentoski, G. D. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations," *International Journal of Robotics Research*, vol. 34, no. 2, pp. 131–157, 2015.

[13] J. Butterfield, S. Osentoski, G. Jay, and O. Jenkins, "Learning from demonstration using a multi-valued function regressor for time-series data," in *IEEE-RAS Int. Conf. Humanoid Robots*, 2010.

[14] G. Chang and D. Kulić, "Robot task learning from demonstration using petri nets," in *IEEE Int. Symp. Robot and Human Interactive Communication*, 2013.

[15] G. Maeda, M. Ewerton, R. Lioutikov, H. Amor, J. Peters, and G. Neumann, "Learning interaction for collaborative tasks with probabilistic movement primitives," in *Int. Conf. Humanoid Robots*, 2014.

[16] J. Kober, M. Gienger, and J. Steil, "Learning movement primitives for force interaction tasks," in *IEEE Int. Conf. Robotics and Automation*, 2015.

[17] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning to sequence movement primitives from demonstrations," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2014.

[18] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[19] J. Zhu and T. Hastie, "Kernel logistic regression and the import vector machine," in *Journal of Computational and Graphical Statistics*, pp. 1081–1088, MIT Press, 2001.

[20] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.